

# An Adaptive Mechanism for Accurate Query Answering under Differential Privacy

Chao Li

University of Massachusetts  
Amherst, Massachusetts, USA  
chaoli@cs.umass.edu

Gerome Miklau

University of Massachusetts  
Amherst, Massachusetts, USA  
miklau@cs.umass.edu

## ABSTRACT

We propose a novel mechanism for answering sets of counting queries under differential privacy. Given a workload of counting queries, the mechanism automatically selects a different set of “strategy” queries to answer privately, using those answers to derive answers to the workload. The main algorithm proposed in this paper approximates the optimal strategy for any workload of linear counting queries. With no cost to the privacy guarantee, the mechanism improves significantly on prior approaches and achieves near-optimal error for many workloads, when applied under  $(\epsilon, \delta)$ -differential privacy. The result is an adaptive mechanism which can help users achieve good utility without requiring that they reason carefully about the best formulation of their task.

## 1. INTRODUCTION

Differential privacy [10] guarantees that information released about participants in a data set will be virtually indistinguishable whether or not their personal data is included. There are now many algorithms satisfying differential privacy [8], however, when adopting differential privacy, users must reason carefully about alternative mechanisms and the formulation of their task. Their choices may have a significant impact on the utility of the output, for the same level of privacy. Even using the PINQ framework [16], designed to aid uninitiated users in writing differentially-private programs, users can be faced with vastly different degrees of accuracy depending on how their task is expressed.

Further, there are few results showing that proposed algorithms are optimally accurate—that is, that they introduce the least possible distortion required to satisfy the privacy criterion.<sup>1</sup> Thus, if the utility they achieve is unacceptable, users often do not know if better utility is possible with a

different algorithm, or if their utility goals are fundamentally incompatible with differential privacy.

In this work, we attempt to relieve the user of some of these difficulties by developing a mechanism that automatically adapts to the set of submitted queries and provides significantly improved utility over competing approaches. We focus on batch query answering, in which a set of queries is answered at one time, in a single interaction with the private server. We call the set of queries a *workload*, which we allow to be any collection of linear counting queries. This general class of queries can be used to express histograms, marginals, data cubes, empirical cumulative distribution functions, common aggregation queries with grouping, and more.

One of the motivations for considering batch query-answering of large workloads is to avoid the complications of online mechanisms in which a user must carefully manage their privacy budget, and, in addition, multiple users may be required to share a single privacy budget to avoid a breach of the privacy definition resulting from collusion. It is therefore appealing to structure large workloads that contain the sufficient statistics of a data mining task, or which can simultaneously support the intended tasks of a group of users. In fact, the output of our algorithms can often be treated as a synthetic data set, albeit one which is tailored specifically for accuracy on the queries in the given workload.

The standard approach for answering a workload of queries under  $\epsilon$ -differential privacy is the Laplace mechanism, which adds to each query a sample chosen independently at random from a Laplace distribution. The noise distribution is scaled to the sensitivity of the workload: the maximum possible change to the query answers induced by the addition or removal of one tuple. Large workloads often have high sensitivity, in which case the Laplace mechanism results in extremely noisy query answers because the noise added to *each* query in the workload is proportional to the sensitivity of the workload.

Recently, a number of related approaches have been proposed which improve on the Laplace mechanism, sometimes allowing for low error where only unacceptably high error was possible before. They each embody a basic (but perhaps counter-intuitive) principle: better results are possible when you *don't ask for what you want*.

The earliest example of this approach focuses on workloads consisting of sets of  $k$ -way marginals, for which Barak et al. answer a set of Fourier basis queries using the Laplace mechanism, and then derive the desired marginals [4]. For workloads consisting of all range-count queries over an or-

<sup>1</sup>For a single numerical query, the addition of appropriately-scaled discrete Laplace noise satisfies  $\epsilon$ -differential privacy and has been proven optimally accurate [11]. For workloads of multiple queries, optimally accurate mechanisms are not known.

dered domain, two approaches have been proposed. Xiao et al. [21] first answer a set of wavelet basis queries, while Hay et al. [13] use a hierarchical set of counting queries which recursively decompose the domain. For workloads consisting of sets of marginals, Ding et al. [7] recently proposed a method for selecting an alternative set of marginals, from which the desired counts can be derived.

These techniques can each be described in the framework of the recently-proposed matrix mechanism [14]. Given a workload of queries, the matrix mechanism uses the Laplace mechanism to answer a set of *strategy* queries. The answers to the strategy queries are then used to derive answers to the workload queries by finding a solution that minimizes squared error. (The derivation by least squares is implicit in Barak [4] and Xiao [21], but explicit in Hay [13] and Ding [7]). In these terms, the four approaches described above can each be seen as providing a set of strategy queries suitable for a particular kind of workload. Ultimately, the use of the strategy queries and the derivation process result in a more complex, non-independent noise distribution which can reduce error.

The matrix mechanism makes clear that nearly any set of strategy queries can be used in this manner to answer a workload. Effective strategies have lower sensitivity than the workload, and are such that the workload queries can be concisely represented in terms of the strategy queries. But the approach remains limited to specific strategies for range queries [13, 21], and approaches which provide only limited choices of strategies for marginals [4, 7].

We continue this line of work in order to create a truly adaptive mechanism that can answer a wide range of workloads with low error. The key to such a mechanism is *strategy selection*: the problem of computing the set of strategy queries that minimizes error for a given workload. Unfortunately, exact solutions to the strategy selection problem are infeasible in practice [14]. One of our main contributions is an approximation algorithm capable of efficiently computing a nearly optimal strategy in  $O(n^4)$  time (where  $n$  is the number of individual counting queries required to express the workload). The result is a mechanism that adapts the noise distribution to the set of queries of interest, relieving the user of the burden of choosing among mechanisms or carefully analyzing their workload.

A few main insights underlie our contributions. First, we shift our focus to  $(\epsilon, \delta)$ -differential privacy, a modest relaxation of  $\epsilon$ -differential privacy. The standard mechanism in this case is the Gaussian mechanism, which suffers the same limitations of the Laplace mechanism, and is also improved by the same approaches described above. The important difference for our results is that sensitivity is measured using the  $L_2$  metric (instead of  $L_1$ ) which ultimately allows for better approximate solutions.<sup>2</sup> Second, inspired by the statistical problem of optimal experimental design [5, 18], we formulate the strategy selection problem as a convex optimization problem which chooses  $n$  coefficients to serve as weights for a fixed set of *design queries*. Third, we show that the eigenvectors of the workload (when represented in matrix form) capture the essential building blocks required for near-optimal strategies and are therefore a very effective

choice for the design queries underlying the above optimization problem.

Our adaptive mechanism advances the state-of-the-art in terms of accuracy, under both absolute and relative measures of error:

- For workloads targeted by prior approaches, our algorithm automatically computes strategies with uniformly lower error. For marginals, our error can be reduced by as much as 6.2 times over Barak and 3.2 times over Ding. For range queries, our error is reduced as much as 2.6 over Xiao and 2.7 times over Hay.
- The power of our adaptive approach is most obvious when applying the mechanism to ad hoc workloads (which may result from specializing a larger workload to a given task, or by combining workloads from multiple users). Error is reduced by as much as 13 times over alternative techniques.
- Our algorithm has a provable approximation ratio and produces strategies with near optimal absolute error for many workloads of interest. We never witness an approximation rate greater than 1.3 times the optimal absolute error. For workloads of marginals, error rates consistently match the optimal achievable error rates.

Our mechanism is also significantly more general than prior work. It can be applied to any workload of linear counting queries: a much larger class of queries than marginals or range queries. In addition, the algorithm avoids a subtle limitation of some previous approaches [13, 21, 7] in which achieving promised error rates depends on finding a proper representation for the workload.

Throughout the paper, all improvements to accuracy are made with *absolutely no cost to privacy*: accuracy is improved by constructing a better noise distribution satisfying the same privacy condition. In addition, while strategy selection is the most computationally intensive part of the mechanism, it only needs to be performed once for any workload, and need not be recomputed to re-run the mechanism on a new database instance. Once the selected strategy is preprocessed, the complexity of executing the mechanism is no higher than applying the standard Laplace mechanism to the workload.

The paper is organized as follows. We review definitions and formally describe the matrix mechanism in Sec. 2. Our algorithm is presented in Sec. 3, along with a theoretical analysis that establishes the approximation rate and other properties. In Sec. 4 we propose performance optimizations which significantly improve computation time with minimal impact on solution quality. In Sec. 5, we evaluate both absolute and relative error rates of our mechanism on a range of workloads. We discuss related work and conclude in Sec. 6 and Sec. 7.

## 2. BACKGROUND

In this section we describe our data model and privacy conditions. We also review the fundamentals of the matrix mechanism, including error measurement and the problem of strategy selection. Throughout the paper, we use the notation of linear algebra and employ standard techniques of matrix analysis. For a matrix  $\mathbf{A}$ ,  $\mathbf{A}^T$  is its transpose and  $\text{trace}(\mathbf{A})$  is the sum of values on the main diagonal. If  $\mathbf{A}$  is a square matrix with full rank,  $\mathbf{A}^{-1}$  denotes its inverse. We

<sup>2</sup>Our algorithm can also be adapted to  $\epsilon$ -differential privacy, but it is less efficient, appears to be less effective, and is significantly harder to analyze. (Please see Sec. 3.5.)

use  $\text{diag}(c_1, \dots, c_n)$  to indicate an  $n \times n$  diagonal matrix with scalars  $c_i$  on the diagonal.

## 2.1 Data Model, Linear Queries, and Workloads

The workloads considered in this paper consist of counting queries over a single relation. Let the database  $I$  be an instance of a single-relation schema  $R(\mathbb{A})$ , with attributes  $\mathbb{A} = \{A_1, A_2, \dots, A_k\}$ . The crossproduct of the attribute domains, written  $\text{dom}(\mathbb{A})$ , is the set of all possible tuples that may occur in  $I$ .

In order to express our queries, we first transform the instance  $I$  into a *data vector*  $\mathbf{x}$  of cell counts. We may choose to fully represent instance  $I$  by defining the vector  $\mathbf{x}$  with one cell for every element of  $\text{dom}(\mathbb{A})$ . Then  $\mathbf{x}$  is a bit vector of size  $|\text{dom}(\mathbb{A})|$  with nonzero counts for each tuple present in  $I$ . This is often inefficient (the size of the  $\mathbf{x}$  vector is the product of the attribute domain sizes) and ineffective (the base counts are typically too small to be estimated accurately under the privacy condition). A common way to form a vector of base counts over larger cells is to partition each  $\text{dom}(A_i)$  into  $d_i$  regions, which could correspond to ranges over an ordered domain or individual elements (or sets of elements) in a categorical domain. Then the individual cells are defined by taking the cross-product of the regions in each attribute. The choice of cells in the data vector is ultimately determined by the workload queries that need to be expressed.

To formally define the data vector we associate, with each element  $x_i$  of  $\mathbf{x}$ , a Boolean *cell condition*  $\phi_i$ , which evaluates to True or False for any tuple in  $\text{dom}(\mathbb{A})$ . We always require that the cell conditions be pairwise unsatisfiable: any tuple in  $\text{dom}(\mathbb{A})$  will satisfy exactly one cell condition. Then  $x_i$  is defined to be the count of the tuples from  $I$  which satisfy  $\phi_i$ .

**DEFINITION 1 (DATA VECTOR).** *Given an ordered list of cell conditions  $\phi_1, \phi_2 \dots \phi_n$  the data vector  $\mathbf{x}$  is a length- $n$  column vector defined by  $n$  positive integral counts  $x_i = |\{t \in I \mid \phi_i(t) \text{ is True}\}|$ .*

In the sequel, the length of  $\mathbf{x}$  is a key parameter, always denoted by  $n$ .

**EXAMPLE 1.** *Consider the relational schema  $R(\text{name}, \text{gradyear}, \text{gender}, \text{gpa})$  describing students. If we wish to form queries only over gender (Male or Female), and gpa ranges  $[1.0, 2.0)$ ,  $[2.0, 3.0)$ ,  $[3.0, 3.5)$ ,  $[3.5, 4.0)$ , then we can define the 8 cell conditions enumerated in Fig. 1(a).*

A linear query computes a specified linear combination of the elements of the data vector  $\mathbf{x}$ .

**DEFINITION 2 (LINEAR QUERY).** *A linear query is a length- $n$  row vector  $\mathbf{q} = [q_1 \dots q_n]$  with each  $q_i \in \mathbb{R}$ . The answer to a linear query  $\mathbf{q}$  on  $\mathbf{x}$  is the vector product  $\mathbf{q}\mathbf{x} = q_1x_1 + \dots + q_nx_n$ .*

In addition to basic predicate counting queries, other aggregates like sum and average, as well as group-by queries, can be expressed as linear counting queries. A *workload* is a set of linear queries. A workload is represented as a matrix, where each row is a single linear counting query.

**DEFINITION 3 (QUERY MATRIX).** *A query matrix is a collection of  $m$  linear queries, arranged by rows to form an  $m \times n$  matrix.*

If  $\mathbf{W}$  is an  $m \times n$  query matrix, the query answer for  $\mathbf{W}$  is a length  $m$  column vector of query results, which can be computed as the matrix product  $\mathbf{W}\mathbf{x}$ . Note that cell condition  $\phi_i$  defines the meaning of the  $i^{\text{th}}$  position of  $\mathbf{x}$ , and accordingly, it determines the meaning of the  $i^{\text{th}}$  column of a query matrix.

**EXAMPLE 2.** *Fig. 1(b) shows a query matrix representing a workload of 8 linear queries. Fig. 1(c) describes the meaning of the queries w.r.t. the cell conditions in Fig. 1(a).*

Note that the data analyst should include in the workload *all* queries of interest, even if some queries could be computed from others in the workload. In the absence of noise introduced by the privacy mechanism, it might be reasonable for the analyst to request answers to a small set of counting queries, from which other queries of interest could be computed. (E.g., it would be sufficient to recover  $\mathbf{x}$  itself by choosing the workload defined by the identity matrix.) But because the analyst will receive private, noisy estimates to the workload queries, the error of queries computed from their combination is often increased. Our adaptive mechanism is designed to optimize error across the entire set of desired queries, so all queries should be included. As a concrete example, in Fig. 1(b),  $\mathbf{q}_3$  can be computed as  $(\mathbf{q}_1 - \mathbf{q}_2)$  but is nevertheless included in the workload.

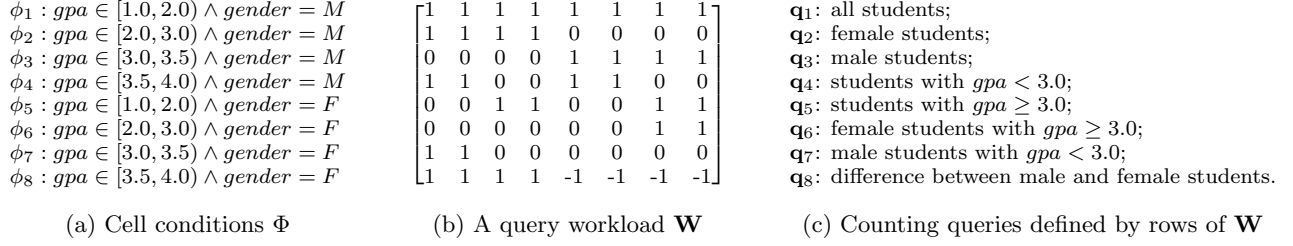
We introduce terminology for a few common workloads used throughout the paper. The relevant properties of workloads are reflected by their matrix representation, so we often drop explicit mention of the schema and attributes involved and focus simply on the number of distinct attributes and the number of disjoint buckets for each attribute, assuming that cells are formed uniformly in the manner described above.

We consider *predicate queries*, *range queries* and *k-way marginal queries*. In addition, since each *k-way marginal query* covers a single value on the margin, one may need to sum answers to multiple marginal queries in order to answer any range query on the margin. When the answers to the marginal queries have noise, summing introduces too much accumulated noise. Therefore, in this paper, we also consider *k-way range marginal queries*, each of which aggregates multiple *k-way marginal queries* so as to cover a range on the margin.

**EXAMPLE 3.** *Of the queries in Fig. 1, the first seven are range queries (and therefore predicate queries as well).  $\mathbf{q}_1 \dots \mathbf{q}_5$  are one-way range marginal queries, in which  $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3$  are one-way range marginal queries over gender and  $\mathbf{q}_1, \mathbf{q}_4, \mathbf{q}_5$  are one-way range marginal queries over gpa;  $\mathbf{q}_2, \mathbf{q}_3$  are also one-way marginal queries.*

We often consider large workloads consisting of *all* queries of a given type, such as “all predicate”, “all range”, “all *k-way range marginal*”, “all *k-way marginal*” and “all marginal” (the union of all *k-way marginals* for  $0 \leq k \leq m$ ). Notice there is no workload of “all range marginal” since it is equivalent to “all range”. Later we will also consider *ad hoc* workloads consisting of arbitrary subsets of each of these types of queries and their combinations. In practice such workloads are important because they may arise from combining queries of interest to multiple users, or from specializing a general workload to a more specific task, to improve error.

## 2.2 Differential Privacy and Gaussian Noise



**Figure 1:** For schema  $R = (name, gradyear, gender, gpa)$ , (a) shows 8 cell conditions on attributes *gender* and *gpa*. The database vector  $\mathbf{x}$  (not shown) will accordingly consist of 8 counts; (b) shows a sample workload matrix  $\mathbf{W}$  consisting of 8 queries, each described in (c).

Standard  $\epsilon$ -differential privacy [10] places a bound (controlled by  $\epsilon$ ) on the difference in the probability of query answers for any two *neighboring* databases. For database instance  $I$ , we denote by  $nbrs(I)$  the set of databases differing from  $I$  in at most one record. Approximate differential privacy [9, 17], is a modest relaxation in which the  $\epsilon$  bound on query answer probabilities may be violated with small probability (controlled by  $\delta$ ).

**DEFINITION 4. (APPROXIMATE DIFFERENTIAL PRIVACY)** A randomized algorithm  $\mathcal{K}$  is  $(\epsilon, \delta)$ -differentially private if for any instance  $I$ , any  $I' \in nbrs(I)$ , and any subset of outputs  $S \subseteq Range(\mathcal{K})$ , the following holds:

$$Pr[\mathcal{K}(I) \in S] \leq \exp(\epsilon) \times Pr[\mathcal{K}(I') \in S] + \delta$$

When  $\delta = 0$ , the condition is standard  $\epsilon$ -differential privacy.

Both definitions can be satisfied by adding random noise to query answers. The magnitude of the required noise is determined by the *sensitivity* of a set of queries: the maximum change in a vector of query answers over any two neighboring databases. But the two privacy definitions differ in the measurement of sensitivity and in their noise distributions. Standard differential privacy can be achieved by adding Laplace noise calibrated to the  $L_1$  sensitivity of the queries [10]. Approximate differential privacy can be achieved by adding Gaussian noise calibrated to the  $L_2$  sensitivity of the queries [9, 17]. Our main results focus on approximate differential privacy, but we discuss extensions to standard differential privacy in Sec 3.5.

Our query workloads are represented as matrices, so we express the sensitivity of a workload as a matrix norm. Because neighboring databases  $I$  and  $I'$  differ in exactly one tuple, and because cell conditions are disjoint, it follows that the corresponding vectors  $\mathbf{x}$  and  $\mathbf{x}'$  differ in exactly one component, by exactly one, in which case we write  $\mathbf{x}' \in nbrs(\mathbf{x})$ . The  $L_2$  sensitivity of  $\mathbf{W}$  is equal to the maximum  $L_2$  norm of the columns of  $\mathbf{W}$ . Below,  $cols(\mathbf{W})$  is the set of column vectors  $W_i$  of  $\mathbf{W}$ .

**PROPOSITION 1 ( $L_2$  QUERY MATRIX SENSITIVITY).** The  $L_2$  sensitivity of a query matrix  $\mathbf{W}$  is denoted  $\|\mathbf{W}\|_2$ , defined as follows:

$$\|\mathbf{W}\|_2 \stackrel{\text{def}}{=} \max_{\mathbf{x}' \in nbrs(\mathbf{x})} \|\mathbf{W}\mathbf{x} - \mathbf{W}\mathbf{x}'\|_2 = \max_{W_i \in cols(\mathbf{W})} \|W_i\|_2$$

For example, for  $\mathbf{W}$  in Fig. 1(b), we have  $\|\mathbf{W}\|_2 = \sqrt{5}$ .

The classic differentially private mechanism adds independent noise calibrated to the sensitivity of a query workload. We use  $Normal(\sigma)^m$  to denote a column vector consisting of  $m$  independent samples drawn from a Gaussian distribution with mean 0 and scale  $\sigma$ .

**PROPOSITION 2. (GAUSSIAN MECHANISM [9, 17])** Given an  $m \times n$  query matrix  $\mathbf{W}$ , the randomized algorithm  $\mathcal{G}$  that outputs the following vector is  $(\epsilon, \delta)$ -differentially private:

$$\mathcal{G}(\mathbf{W}, \mathbf{x}) = \mathbf{W}\mathbf{x} + Normal(\sigma)^m$$

where  $\sigma = \|\mathbf{W}\|_2 \sqrt{2 \ln(2/\delta)}/\epsilon$

Recall that  $\mathbf{W}\mathbf{x}$  is a vector of the true answers to each query in  $\mathbf{W}$ . The algorithm above adds independent Gaussian noise (scaled by  $\epsilon$ ,  $\delta$ , and the sensitivity of  $\mathbf{W}$ ) to each query answer. Thus  $\mathcal{G}(\mathbf{W}, \mathbf{x})$  is a length- $m$  column vector containing a noisy answer for each linear query in  $\mathbf{W}$ .

## 2.3 The Matrix Mechanism

The matrix mechanism has a form similar to the Gaussian mechanism but adds a more complex noise vector. It uses a strategy matrix,  $\mathbf{A}$ , to construct this vector.

**PROPOSITION 3. (( $\epsilon, \delta$ )-MATRIX MECHANISM [14])** Given an  $m \times n$  query matrix  $\mathbf{W}$ , and assuming  $\mathbf{A}$  is a full rank  $p \times n$  strategy matrix, the randomized algorithm  $\mathcal{M}_{\mathbf{A}}$  that outputs the following vector is  $(\epsilon, \delta)$ -differentially private:

$$\mathcal{M}_{\mathbf{A}}(\mathbf{W}, \mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{W}\mathbf{A}^+ Normal(\sigma)^m.$$

where  $\sigma = \|\mathbf{A}\|_2 \sqrt{2 \ln(2/\delta)}/\epsilon$

Here  $\mathbf{A}^+$  is the pseudo-inverse of  $\mathbf{A}$ :  $\mathbf{A}^+ = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$ ; if  $\mathbf{A}$  is a square matrix, then  $\mathbf{A}^+$  is just the inverse of  $\mathbf{A}$ . The intuitive justification for this mechanism is that it is equivalent to the following three-step process: (1) the queries in the strategy are submitted to the Gaussian mechanism; (2) an estimate  $\hat{\mathbf{x}}$  for  $\mathbf{x}$  is derived by computing the  $\hat{\mathbf{x}}$  that minimizes the squared sum of errors (this step consists of standard linear regression and requires that  $\mathbf{A}$  be full rank to ensure a unique solution); (3) noisy answers to the workload queries are then computed as  $\mathbf{W}\hat{\mathbf{x}}$ . The answers to  $\mathbf{W}$  derived in step (3) are always consistent because they are computed from a single noisy version of the cell counts,  $\hat{\mathbf{x}}$ .

Like the Gaussian mechanism, the matrix mechanism computes the true answer vector  $\mathbf{W}\mathbf{x}$  and adds noise to each component. But a key difference is that the scale of the Gaussian noise is calibrated to the *sensitivity of the strategy matrix  $\mathbf{A}$* , not that of the workload. In addition, the noise added to query answers is no longer independent, because the vector of independent Gaussian samples is transformed by the matrix  $\mathbf{W}\mathbf{A}^+$ .

**EXAMPLE 4.** The three strategy matrices in Fig. 2 can be used by the matrix mechanism to answer the workload  $\mathbf{W}$  in Fig. 1(b), with differing results. The first strategy is the identity matrix, the second is the Haar wavelet strategy, and the

Identity	Wavelet
$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$
Adaptive Algorithm Output	
$\begin{bmatrix} 0 & 0 & -.1 & .1 & .19 & -.19 & 0 & 0 \\ 0 & 0 & .19 & -.19 & .1 & -.1 & 0 & 0 \\ -.3 & -.3 & .33 & .33 & .33 & .33 & -.3 & -.3 \\ .47 & .47 & -.5 & -.5 & .5 & .5 & -.47 & -.47 \\ .56 & .56 & .53 & .53 & -.53 & -.53 & .56 & .56 \\ .62 & .62 & .57 & .57 & .57 & .57 & .62 & .62 \end{bmatrix}$	

**Figure 2:** Alternative strategy matrices that can be used to answer workload  $\mathbf{W}$  from Fig 1(b). The root mean square error of answering  $\mathbf{W}$  when using the identity, wavelet, and adapted strategies is 45.36, 34.62 and 29.79, respectively.

third is the output of the algorithm proposed in Sec 3. With  $\epsilon = 0.5$  and  $\delta = 0.0001$ , if the workload itself is used as the strategy, the root mean square error of answering  $\mathbf{W}$  is 47.78. The root mean square error using the identity, wavelet, and adaptive strategies is 45.36, 34.62 and 29.79, respectively. It is possible to prove that no strategy can answer  $\mathbf{W}$  with error less than 29.18, so the algorithm is finding a nearly optimal strategy for this workload.

Intuitively, by using the identity strategy, we get noisy estimates of each cell count using the Gaussian mechanism, and then use those estimates to compute the workload queries. This strategy performs poorly for workload queries that sum many base counts because the variance of the independent noise increases additively. The wavelet addresses this limitation by allowing large range queries to be estimated by combining the answers to just a few of the wavelet strategy queries. It offers a dramatic improvement over the identity strategy for workload consisting of all range queries. However, the wavelet is not necessarily appropriate for every workload. Our algorithm produces a strategy customized to  $\mathbf{W}$ , allowing for reduced error.

## 2.4 Optimal Error for the Matrix Mechanism

We measure the accuracy of a noisy query answer using root mean square error. For a workload of queries, the error is defined as the root mean square error of the vector of answers, which we refer to simply as *workload error* in the remainder of the paper.

**DEFINITION 5 (QUERY AND WORKLOAD ERROR).** Let  $\hat{\mathbf{w}}$  be the estimate for query  $\mathbf{w}$  under the matrix mechanism using query strategy  $\mathbf{A}$ . That is,  $\hat{\mathbf{w}} = \mathcal{M}_{\mathbf{A}}(\mathbf{w}, \mathbf{x})$ . The query error of the estimate for  $\mathbf{w}$  using strategy  $\mathbf{A}$  is:

$$\text{ERROR}_{\mathbf{A}}(\mathbf{w}) \stackrel{\text{def}}{=} \sqrt{\mathbb{E}[(\mathbf{w}\mathbf{x} - \hat{\mathbf{w}})^2]}.$$

Given a workload  $\mathbf{W}$  consisting of  $m$  queries, the workload error of answering  $\mathbf{W}$  using strategy  $\mathbf{A}$  is:

$$\text{ERROR}_{\mathbf{A}}(\mathbf{W}) \stackrel{\text{def}}{=} \sqrt{\frac{1}{m} \sum_{\mathbf{w}_i \in \mathbf{W}} \text{ERROR}_{\mathbf{A}}(\mathbf{w}_i)^2}.$$

The query answers returned by the matrix mechanism are linear combinations of noisy strategy query answers to which independent Gaussian noise has been added. Thus, as the following proposition shows, we can directly compute the error for any linear query  $\mathbf{w}$  or workload  $\mathbf{W}$  as a function of  $\epsilon$ ,  $\delta$ , and  $\mathbf{A}$ :

**PROPOSITION 4. (WORKLOAD ERROR)** Given a workload  $\mathbf{W}$ , the error of answering  $\mathbf{W}$  using the  $(\epsilon, \delta)$  matrix mechanism with query strategy  $\mathbf{A}$  is:

$$\text{ERROR}_{\mathbf{A}}(\mathbf{W}) = \|\mathbf{A}\|_2 \sqrt{P(\epsilon, \delta) \text{trace}(\mathbf{W}^T \mathbf{W} (\mathbf{A}^T \mathbf{A})^{-1})} \quad (1)$$

where  $P(\epsilon, \delta) = \frac{2 \log(2/\delta)}{\epsilon^2}$ .

To build a mechanism that adapts to a given workload  $\mathbf{W}$ , our goal is to select a strategy  $\mathbf{A}$  to minimize the above formula. The optimal strategy for a workload  $\mathbf{W}$  is defined to be one that minimizes the workload error:

**PROBLEM 1. (OPTIMAL STRATEGY SELECTION)** Given a workload  $\mathbf{W}$ , find a query strategy  $\mathbf{A}_0$  such that:

$$\text{ERROR}_{\mathbf{A}_0}(\mathbf{W}) = \min_{\mathbf{A}} \text{ERROR}_{\mathbf{A}}(\mathbf{W}). \quad (2)$$

We denote the problem of computing an optimal strategy matrix as  $\text{OPTSTRAT}(\mathbf{W})$  and the workload error under this strategy as  $\text{OPTERROR}(\mathbf{W})$ . It is possible to compute an exact solution to  $\text{OPTSTRAT}(\mathbf{W})$  by representing it as a convex optimization problem [14]. However, encoding the necessary constraints results in a problem with a large number of variables and optimization takes  $O(n^8)$  time with standard solvers, making it infeasible for practical applications. One of our main goals is to efficiently find approximately optimal strategy matrices, for any provided workload.

We emphasize that the algorithms in this paper optimize the workload error, an *absolute* measure of error. The solution to this optimization problem depends on the workload alone, not on the input database. (This is evident from the fact that  $\mathbf{x}$ , the vector of database counts, does not appear in Eq. (1) above.) We also consider *relative* error in the experimental evaluation, which inherently depends on the input database. We show that low relative error for a workload  $\mathbf{W}$  can be achieved by optimizing (absolute) error of a workload whose rows have been scaled in a straightforward way.

## 3. AN ALGORITHM FOR EFFICIENT STRATEGY SELECTION

In this section we present an approximation algorithm for the strategy selection problem, prove its approximation rate and other properties, and discuss adapting the algorithm to  $\epsilon$ -differential privacy.

### 3.1 Optimal Query Weighting

The main difficulty in solving  $\text{OPTSTRAT}(\mathbf{W})$  is computing (subject to complex constraints) all  $n^2$  entries of a strategy matrix. To simplify the problem, we take inspiration from the related problem of *optimal experimental design* [18].

Consider a scientist who wishes to estimate the value of  $n$  unknown variables as accurately as possible. The variables cannot be observed directly, but only by running one or more of a fixed set of feasible experiments, each of which returns a linear combination of the variables. The experiments suffer from observational error, but those errors are assumed independent, and it follows that the least square method can be used to estimate the unknown variables once the results of the experiments are collected. Each experiment has an associated cost (which may represent time, effort, or financial expense) and the scientist has a fixed budget. The optimal experimental design is the subset (or weighted subset) of feasible experiments offering the best estimate of the unknown variables and with a cost less than the budget constraint.

There is an immediate analogy to the problem of strategy selection: our strategy queries are like experiments that provide partial information about the unknown data vector  $\mathbf{x}$ , and the final result will be computed using the least square method. However, in our setting, we are permitted to ask any query, with a cost (arising from the increase in sensitivity) which impacts the added noise. In addition, our goal is to minimize the workload error, while experimental design always minimizes the error of the individual variables (i.e. the error metric in experimental design is equivalent to our problem only if  $\mathbf{W}$  is the identity matrix).

Despite these important differences, we adopt from experimental design the idea to limit the selection of our strategy to weighted combinations of a set of *design queries* that are fixed ahead of time. Naturally, design queries with a weight of zero are omitted. For a set of design queries  $\mathcal{Q}$ , the following problem, denoted  $\text{OPTSTRAT}_{\mathcal{Q}}(\mathbf{W})$ , selects the set of weights which minimizes the workload error for  $\mathbf{W}$ .

**PROBLEM 2 (APPROXIMATE STRATEGY SELECTION).**

Let  $\mathbf{W}$  be a workload and  $\mathcal{Q} = \{\mathbf{q}_1, \dots, \mathbf{q}_k\}$  the design queries. For weights  $\Lambda = (\lambda_1 \dots \lambda_k) \in \mathbb{R}^k$ , let matrix  $\mathbf{A}_{\Lambda, \mathcal{Q}} = [\lambda_1 \mathbf{q}_1, \dots, \lambda_k \mathbf{q}_k]^T$ . Choose weights  $\Lambda_0 \in \mathbb{R}^k$  such that:

$$\text{ERROR}_{\mathbf{A}_{\Lambda_0, \mathcal{Q}}}(\mathbf{W}) = \min_{\Lambda \in \mathbb{R}^k} \text{ERROR}_{\mathbf{A}_{\Lambda, \mathcal{Q}}}(\mathbf{W}). \quad (3)$$

The solution to this problem only approximates the truly optimal strategy since it is limited to selecting a strategy that is a weighted combination of the design queries. But  $\text{OPTSTRAT}_{\mathcal{Q}}(\mathbf{W})$  can be computed much more efficiently than  $\text{OPTSTRAT}(\mathbf{W})$ . To do so, we describe  $\text{OPTSTRAT}_{\mathcal{Q}}(\mathbf{W})$  as a semi-definite program [5], a special form of convex optimization in which a linear objective function is minimized over the cone of positive semidefinite matrices. Below,  $\circ$  is the Hadamard (entry-wise) product of two matrices, and for symmetric matrix  $\mathbf{Q}$ ,  $\mathbf{Q} \succeq 0$  denotes that  $\mathbf{Q}$  is positive semidefinite, which means  $\mathbf{x}^T \mathbf{Q} \mathbf{x} \geq 0$  for any vector  $\mathbf{x}$ .

---

#### Program 1 Optimal Query Weighting

---

**Given:**  $c_1, \dots, c_n$ ,  $\mathcal{Q} = [\mathbf{q}_1, \dots, \mathbf{q}_n]$ .  
**Choose:**  $u_1, \dots, u_n$ ,  $v_1, \dots, v_n$ .  
**Minimize:**  $c_1 v_1 + \dots + c_n v_n$ .  
**Subject to:**  $\begin{bmatrix} u_i & 1 \\ 1 & v_i \end{bmatrix} \succeq 0, \quad i = 1, \dots, n.$   
 $(\mathcal{Q} \circ \mathcal{Q})^T \mathbf{u} \leq \mathbf{1}.$

---

**THEOREM 1.** Given a workload  $\mathbf{W}$  and a set of design queries  $\mathcal{Q} = \{\mathbf{q}_1, \dots, \mathbf{q}_n\}$ , let  $c_1, \dots, c_n$  be the squared  $L_2$  norms of the columns of matrix  $\mathbf{W}\mathcal{Q}^+$ . If the output of Program 1 is  $u_1, \dots, u_n$  then setting  $\Lambda = \{\sqrt{u_1} \dots \sqrt{u_n}\}$  achieves  $\text{OPTSTRAT}_{\mathcal{Q}}(\mathbf{W})$ .

Algorithms for efficiently solving semidefinite programs have received considerable attention recently [5]. Using standard algorithms, Program 1 can be solved in  $O(n|\mathcal{Q}|^3)$  time. Recall that the complexity of computing  $\text{OPTSTRAT}(\mathbf{W})$  is  $O(n^8)$ . Thus, Program 1 offers an efficiency improvement as long as  $|\mathcal{Q}| = O(n^2)$ . This provides a target size for selecting the design set, which we turn to next.

### 3.2 Choosing the Design Queries

The potential of the above approach depends on finding a set of design queries,  $\mathcal{Q}$ , that is concise (containing no

more than  $n^2$ , and preferably  $n$ , queries) and also expressive (so that near-optimal solutions can be expressed as weighted combinations of its elements).

One straightforward idea is to adopt as the design queries one of the proposed strategy matrices from prior work. These are good strategy matrices for specific workloads such as the set of all range queries (wavelet or hierarchical strategy) or sets of low order marginals (the Fourier strategy). Choosing one of these for  $\mathcal{Q}$  would guarantee that  $\text{OPTSTRAT}_{\mathcal{Q}}(\mathbf{W})$  produces a solution that improves upon the error of using that strategy. Unfortunately these strategies are not sufficiently expressive for workloads very different from their target workloads.

Another possibility is to use the workload itself as the set of design queries, but there are two difficulties with this. First, there is no guarantee that a workload includes within it the components from which a high quality strategy may be formed, especially if the workload only contains a small set of queries. The workloads of all range and all predicate queries are in fact sufficiently expressive (e.g. both the hierarchical strategy and a strategy equivalent to wavelet can be constructed by applying weights to the set of all range queries). But this leads to the second issue: these workloads, and others that serve important applications, are too large and fail to meet our conciseness requirement.

To avoid these pitfalls, we will derive the design set from the given workload  $\mathbf{W}$  by applying tools of spectral analysis. Intuitively this is a good choice because the eigenvectors of a matrix often capture its most important properties. We will also show in the next section that this choice aids in the theoretical analysis of the approximation ratio because it allows us to relate the output of  $\text{OPTSTRAT}_{\mathcal{Q}}(\mathbf{W})$  to a lower bound on error that is a function of the workload eigenvalues.

Recall that the key part of the expression for Eqn. (1) in Prop. 4 is  $\text{trace}(\mathbf{W}^T \mathbf{W} (\mathbf{A}^T \mathbf{A})^{-1})$ , and notice that the workload occurs only in the form of  $\mathbf{W}^T \mathbf{W}$ . It follows that there are many workloads with equivalent error because it is easy to construct a matrix  $\mathbf{W}_0$  such that  $\mathbf{W}_0^T \mathbf{W}_0 = \mathbf{W}^T \mathbf{W}$  by letting  $\mathbf{W}_0 = \mathbf{Q}\mathbf{W}$  for any orthogonal matrix  $\mathbf{Q}$ . This suggests that, as far as workload error under the matrix mechanism is concerned, the essential properties of the workload are reflected by  $\mathbf{W}^T \mathbf{W}$ . This motivates the following definition of eigen-queries of a workload, which we will use as our design set.

**DEFINITION 6 (EIGEN-QUERIES OF A WORKLOAD).**

Given a workload  $\mathbf{W}$ , consider the eigen-decomposition of  $\mathbf{W}^T \mathbf{W}$  into  $\mathbf{W}^T \mathbf{W} = \mathbf{Q}^T \mathbf{D} \mathbf{Q}$ , where  $\mathbf{Q}$  is an orthogonal matrix and  $\mathbf{D}$  is a diagonal matrix. The *eigen-queries* of  $\mathbf{W}$  are the rows of  $\mathbf{Q}$  (i.e. the eigenvectors of  $\mathbf{W}^T \mathbf{W}$ ).

Choosing the eigen-queries of  $\mathbf{W}$  as the design set meets our conciseness requirement because there are never more than  $n$  eigen-queries. Thus Program 1,  $\text{OPTSTRAT}_{\mathcal{Q}}(\mathbf{W})$ , has complexity  $O(n^4)$ , which is  $O(n^4)$  times faster than solving  $\text{OPTSTRAT}(\mathbf{W})$ . We also find that the eigen-queries meet our expressiveness objective. We will show this next by proving a bound on the approximation ratio. In Sec. 4 we propose techniques that exploit the fact that using subsets of the eigen-queries retain much of the expressiveness and increase efficiency. And in Section 5, we show experimentally that weighted eigen-queries allow for near-optimal strategies, and also that the eigen-queries outperform other natural alternatives for the design set.

### 3.3 The Eigen-Design Algorithm

It remains to define the complete Eigen-Design algorithm, which is Program 2:

---

#### Program 2 The Eigen-Design Algorithm

---

**Input:** Workload matrix  $\mathbf{W}$ .

**Output:** Strategy matrix  $\mathbf{A}$ .

- 1: Compute the eigenvalue decomposition of  $\mathbf{W}^T \mathbf{W} = \mathbf{Q}^T \mathbf{D} \mathbf{Q}$ , where  $\mathbf{D} = \text{diag}(\sigma_1, \dots, \sigma_n)$  and set  $\mathcal{Q} = \mathbf{Q}$ .
  - 2: Compute weights  $\lambda_1, \dots, \lambda_n$  by solving Program 1 for above  $\mathcal{Q}$  and with  $c_i = \sigma_i$ ,  $i \in [1..n]$ .
  - 3: Construct matrix  $\mathbf{A}' = \mathbf{A} \mathbf{Q}$  where  $\mathbf{A} = \text{diag}(\lambda_1, \dots, \lambda_n)$ .
  - 4: Let  $m_{11}, \dots, m_{nn}$  be the  $L_2$  norm of columns of  $\mathbf{A}'$  and define  $\mathbf{D}' = \text{diag}(\max_i \{\sqrt{m_{ii}^2 - m_{11}^2}\}, \dots, \max_i \{\sqrt{m_{ii}^2 - m_{nn}^2}\})$ .
  - 5: Return  $\mathbf{A} = \begin{bmatrix} \mathbf{A}' \\ \mathbf{D}' \end{bmatrix}$ .
- 

The algorithm performs the decomposition of  $\mathbf{W}^T \mathbf{W}$  to derive the design queries (Step 1), and solves  $\text{OPTSTRAT}_{\mathcal{Q}}(\mathbf{W})$  using the eigen-queries as the design set (Step 2). The matrix  $\mathbf{A}'$  that is constructed in Step 3 is a candidate strategy but may have one or more columns whose norm is less than the sensitivity. In this case, it is possible to add queries, completing columns, without raising the sensitivity (Step 4 and 5). These additional queries can only provide more information about the database, and hence reduce error.

### 3.4 Analysis of the Eigen-Design Algorithm

We now consider the accuracy and generality of the eigen-design algorithm, showing a bound on the worst-case approximation rate and that the accuracy of the algorithm is robust with respect to the representation of the input workload.

#### Approximation Rate

To bound the approximation rate, we use an existing result showing a lower bound on the optimal error achievable for a workload using the  $(\epsilon, \delta)$ -matrix mechanism [15]. The existence of this bound does not imply an algorithm for achieving it, but it is a useful tool for understanding theoretically and experimentally the quality of the strategies produced by  $\text{OPTSTRAT}(\mathbf{W})$  using the eigenvalues of  $\mathbf{W}$ .

**THEOREM 2. (SINGULAR VALUE BOUND [15])** *Given any  $m \times n$  workload  $\mathbf{W}$ . Let  $\sigma_1, \dots, \sigma_n$  be the eigenvalues of matrix  $\mathbf{W}^T \mathbf{W}$ . The singular value bound of  $\mathbf{W}$  is  $\text{SVDB}(\mathbf{W}) = \frac{1}{n}(\sqrt{\sigma_1} + \dots + \sqrt{\sigma_n})^2$ , and bounds  $\text{OPTERROR}(\mathbf{W})$ :*

$$\text{OPTERROR}(\mathbf{W}) \geq \sqrt{P(\epsilon, \delta) \text{SVDB}(\mathbf{W})}.$$

Intuitively, let  $\mathbf{A}_l$  be the strategy that is defined by weighting the eigen queries of  $\mathbf{W}$  by  $\sqrt{\sigma_1}, \dots, \sqrt{\sigma_n}$ . The singular value bound comes from underestimating the sensitivity of  $\mathbf{A}_l$  using  $\sqrt{\text{trace}(\mathbf{A}_l^T \mathbf{A}_l)}/n$ . In practice, though the singular value bound may not be achieved since there is a gap between the sensitivity of  $\mathbf{A}_l$  and  $\sqrt{\text{trace}(\mathbf{A}_l^T \mathbf{A}_l)}/n$ , the idea of weighting the eigen queries can be combined with the experimental design method to find good strategies to  $\mathbf{W}$ .

Notice the strategy  $\mathbf{A}_l$  is contained in the possible solutions of Program 2. Thus the approximation ratio of Program 2 can be estimated by using the approximation ratio of the singular value bound.

**THEOREM 3.** *Given workload  $\mathbf{W}$ , let  $\sigma_1$  be the largest eigenvalue of  $\mathbf{W}^T \mathbf{W}$ , Program 2 gives a strategy that approximates  $\text{OPTERROR}(\mathbf{W})$  with a ratio of  $(n\sigma_1/\text{SVDB}(\mathbf{W}))^{1/4}$ .*

This theorem shows that the approximation ratio of applying Program 2 to a workload  $\mathbf{W}$  can be bounded by analyzing the eigenvalues of matrix  $\mathbf{W}^T \mathbf{W}$ .

In practice, the ratio between the error of the eigen strategies and the optimal error is much smaller for a wide range of common workloads. In the experiments in Sec. 5, the largest ratio is at most 1.3 and in a number of cases the ratio is essentially equal to 1, modulo numerical imprecision.

#### Representation Independence

We say that the Eigen-Design algorithm is representation independent because its output is invariant for semantically equivalent workloads and error equivalent workloads. Recall that the logical semantics of a workload matrix  $\mathbf{W}$  depends on its cell conditions. For any workload matrix  $\mathbf{W}$ , reordering its cell conditions leads to a new matrix  $\mathbf{W}'$  with accordingly reordered columns. In this case, we say  $\mathbf{W}$  and  $\mathbf{W}'$  are *semantically-equivalent*.

Naturally, we hope for a mechanism with equal error for any two semantically-equivalent representations of a workload. Some prior approaches do not have this property. For example, the wavelet and hierarchical strategies exploit the locality present in the canonical representation of range queries. An alternative matrix representation of the range queries may result in significantly larger error. The Eigen-Design algorithm does not suffer from this pitfall:

**PROPOSITION 5 (SEMANTIC EQUIVALENCE).** *Let  $\mathbf{W}_1$  and  $\mathbf{W}_2$  be two semantically-equivalent workloads and suppose Prog. 2 computes strategy  $\mathbf{A}_1$  on workload  $\mathbf{W}_1$  and  $\mathbf{A}_2$  on workload  $\mathbf{W}_2$ . Then  $\text{ERROR}_{\mathbf{A}_1}(\mathbf{W}_1) = \text{ERROR}_{\mathbf{A}_2}(\mathbf{W}_2)$ .*

A related issue arises for two workloads that may be semantically different, but can be shown to have equivalent error. Since  $\mathbf{W}$  appears as  $\mathbf{W}^T \mathbf{W}$  in the expression for error of a workload, it follows that, for any orthogonal matrix  $\mathbf{Q}$ , workload  $\mathbf{Q} \mathbf{W}$  has error equal to  $\mathbf{W}$  under any strategy. And in particular, any two such workloads have equal minimum error. The Eigen-Design algorithm always finds the same strategies for any two error-equivalent workloads:

**PROPOSITION 6 (ERROR EQUIVALENCE).** *Let  $\mathbf{W}_1$  and  $\mathbf{W}_2$  be two error-equivalent workloads (i.e.  $\mathbf{W}_1 = \mathbf{Q} \mathbf{W}_2$  for some orthogonal  $\mathbf{Q}$ ) and suppose Program 2 computes strategy  $\mathbf{A}_1$  on workload  $\mathbf{W}_1$  and  $\mathbf{A}_2$  on workload  $\mathbf{W}_2$ . Then  $\text{ERROR}_{\mathbf{A}_1}(\mathbf{W}_1) = \text{ERROR}_{\mathbf{A}_2}(\mathbf{W}_2)$ .*

This result follows from the fact that the input to Program 1 uses the eigenvectors of  $\mathbf{W}^T \mathbf{W}$ , and therefore operates identically on equivalent workloads.

#### Optimizing for Relative Error

The discussion above is about workload error, an *absolute* measure of error. Our adaptive approach can also be used to find strategies offering low *relative* error. However, these are two fundamentally different optimization objectives and a single strategy matrix will not, in general, satisfy both.

One major difference between computing absolute error and relative error is the impact of the  $L_2$  norm of a query vector. According to Prop. 3 and Def. 5, the query error of  $\mathbf{w}$  under strategy  $\mathbf{A}$  is proportional to the  $L_2$  norm of  $\mathbf{w}$ . Therefore a scaled query  $k\mathbf{w}$  has  $k$  times larger query error compared with  $\mathbf{w}$ , and thus a query with higher  $L_2$  norm contributes more to workload error. But because the

relative error does not change with the  $L_2$  norm of the query, using strategies optimized for workload error will not lead to optimal relative error.

Because the matrix mechanism is a data-independent mechanism, it is not possible to optimize for relative error directly. If the distribution of the target dataset were known, we could scale each query by its weighted  $L_2$  norm, where the weight on each cell is proportional to the inverse of its probability. This scaling will optimize towards relative error by neutralizing the fact that the designed strategies are biased towards high norm queries. Since the underlying distribution is typically unknown, we introduce a heuristic scaling, prior to applying the Eigen-Design algorithm, in which each query is normalized to make its  $L_2$  norm 1. This is equivalent to assuming a uniform distribution over the cells. In Sec 5, we show that, for two real datasets, this approach results in significantly lower relative error than competing techniques.

### 3.5 Application to the $\epsilon$ -Matrix Mechanism

There are a number of challenges to applying the optimally weighted design approach under  $\epsilon$ -differential privacy. Recall, once again, the formula for workload error from Prop. 4:  $\|\mathbf{A}\|_2 \sqrt{\text{trace}(\mathbf{W}^T \mathbf{W} (\mathbf{A}^T \mathbf{A})^{-1})}$ . To move to  $\epsilon$ -differential privacy, only the sensitivity term changes, from  $L_2$  to  $L_1$ :  $\|\mathbf{A}\|_1 \sqrt{\text{trace}(\mathbf{W}^T \mathbf{W} (\mathbf{A}^T \mathbf{A})^{-1})}$ . In the former case, the sensitivity term  $\|\mathbf{A}\|_2$  is uniquely determined by  $\mathbf{A}^T \mathbf{A}$ . But in the latter case, computing a near-optimal  $\mathbf{A}^T \mathbf{A}$  is not enough, because  $\|\mathbf{A}\|_1$  remains undetermined and is itself hard to optimize. As a result, it is more challenging (although still possible) to represent the optimal query weighting as a convex optimization problem. We omit its formal encoding, but note that the resulting problem is also less efficient because we can no longer rely on second order cone programming.

Furthermore, there does not seem to be a universally good design set: the eigen-queries do not outperform other bases, in general, because they characterize only the properties of  $\mathbf{W}^T \mathbf{W}$  but do not account for the  $L_1$  sensitivity. We can nevertheless still use our algorithm to improve existing strategies. For example, using the Wavelet basis in the algorithm can improve its performance on all range and random range queries by a factor of 1.1 and 1.5, respectively; using the Fourier basis can improve its performance on low order marginals by a factor of 1.6.

Lastly, we do not know of an analogue of Thm 2 providing a guaranteed error bound for the  $\epsilon$ -Matrix Mechanism to verify the quality of the output.

These challenges motivate our choice to focus on  $(\epsilon, \delta)$ -differential privacy. While the two privacy guarantees are strictly-speaking incomparable, for conservative settings of  $\delta$ , a user may be indifferent between the two. It is then possible to show that the asymptotic error rates for many workloads are roughly comparable between the two models.

## 4. COMPLEXITY AND OPTIMIZATIONS

We focus next on methods to further reduce the complexity of approximate strategy selection. We first analyze the complexity of the strategy selection algorithm and show that it can be solved more efficiently for low rank workloads, with no impact on the quality of the solution. Then we propose two approaches which can significantly speed up strategy selection by reducing the size of the input to Program 2. Intuitively, both approaches perform strategy selection over a

summary of the workload that is constructed from its most significant eigenvectors, potentially sacrificing fidelity of the solution. We evaluate the latter two techniques in Sec 5.2.

### 4.1 Complexity Analysis

The rank of workload matrix  $\mathbf{W}$ , denoted by  $\text{rank}(\mathbf{W})$ , is the size of the largest linearly-independent subset of the rows (or, equivalently, columns). When  $\text{rank}(\mathbf{W})$  is its maximum value,  $n$ , we say that  $\mathbf{W}$  has full rank, which implies that accurate answers to the workload queries in  $\mathbf{W}$  uniquely determine every cell count in  $\mathbf{x}$ . The complexity of the strategy selection algorithm can be broken into three parts: computing the eigenvectors and eigenvalues of matrix  $\mathbf{W}^T \mathbf{W}$ , solving the optimization problem, and constructing the strategy. If an eigenvalue is equal to zero, the eigenvalue and its corresponding eigenvectors are not actually involved in the optimization and strategy construction, so they can be omitted in practice. Since the number of nonzero eigenvalues of  $\mathbf{W}^T \mathbf{W}$  is equal to  $\text{rank}(\mathbf{W})$ , the complexity of Programs 2 is  $O(nm \text{rank}(\mathbf{W}) + n \text{rank}(\mathbf{W})^3)$ .

The complexity analysis above indicates that its efficiency can be significantly improved when  $\text{rank}(\mathbf{W}) \ll n$ . For example, the rank of low order marginal workloads can be bounded by the number of queries in the workload. Suppose a low-order marginal workload is defined on a  $k$ -dimensional space of cell conditions, each of which has size  $d$ . If the workload only contains one-way marginals, the complexity of solving Program 2 over this workload is bounded by  $O(k^3 d^{3+k})$ . If the workload consists of one and two-way marginals the complexity is  $O(k^6 d^{k+6})$ . Both of these bounds are much smaller than  $O(d^{4k})$ .

### 4.2 Workload Reduction Approaches

Next we propose two approaches which allow us to reduce the number of variables in the optimization problem. Both are inspired by principal component analysis (PCA), in which a matrix is characterized by the so-called principal eigenvectors, which are the eigenvectors associated with the largest eigenvalues.

In our case, recall that we cannot ignore the non-principal eigenvectors since the rank of the strategy matrix  $\mathbf{A}$  cannot be lower than the workload matrix  $\mathbf{W}$ . Instead, we either compute separately the weights for the principal and remaining eigenvectors, or we choose the same weights for all the remaining eigenvectors.

#### *Eigen-Query Separation*

In *eigen-query separation*, we partition the eigen-queries into groups of a specified size according to their corresponding eigenvalues. Treating one group at a time, Program 1 is executed to determine the optimal weights just for the eigenvectors of that group. After the individual group optimizations are finished, another optimization can be used to calculate the best factor to be applied to all queries in each group. If the group size is large, all of the principal eigenvectors may be contained in one group, in which case the most important weights will be computed precisely.

The complexity of eigen-query separation depends on the group division. Notice that during the optimization of each group, the convex optimization problem is equivalent to setting all eigenvalues of excluded eigenvectors to zero. Analogous to the discussion of low rank workloads, letting the size of group be  $n_g$ , the complexity of solving the optimization



problem over each group is  $O(nn_g^3)$ . Similarly, the time complexity to combine all the groups is  $O(n(n/n_g)^3)$ , and therefore  $O(n^2n_g^3 + n(n/n_g)^3)$  in total. Asymptotically, the complexity of eigen-query separation is minimized when  $n_g = O(n^{1/3})$ . Then the complexity of the entire process is  $O(n^3)$ , the same as the cost of standard matrix multiplication.

### Principal Vectors Optimization

In the *principal vector optimization* we use a subset of the  $k$  most important eigenvectors as the design set, computing the optimal weights as usual. Instead of ignoring the less important eigenvectors (as is typical in PCA) we simply use a single common weight for each of the excluded vectors that have non-zero eigenvalues. The number of variables in the convex optimization is reduced to  $k + 1$  so that the time complexity is reduced to  $O(nk^3)$ . Experimentally we find that good results are possible with as little as 10% of the eigenvectors.

In Sec. 5.2 we show that both of the above approaches can improve execution time by two orders of magnitude with modest impact on solution quality. Extending our theoretical bound on the approximation rate to these approaches is an interesting direction for future work.

## 5. EXPERIMENTAL EVALUATION

The empirical evaluation of our mechanism has three objectives: (i.) to measure solution quality of the Eigen-Design algorithm using both absolute and relative error; (ii.) to measure the trade-off between speed-up and solution quality of our two performance optimizations; and (iii.) to measure the effectiveness of using the eigen-queries as the design set. Experimental conclusions are presented in Sec. 5.4.

### Experimental Setup

Recall that workload error is an absolute error measure based on root mean square error. Workload error can be analytically computed using Prop. 4, and this is precisely the error that will be witnessed when running repeated trials and computing the mean deviation. Further, workload error is independent of the true counts in data vector  $\mathbf{x}$ . That is, it is independent of the input data. These facts hold for all instances of the matrix mechanism, and therefore for each of the competing techniques we consider below. Therefore, when evaluating this absolute error measure, we do not perform repeated trials with samples of random noise nor do we use any datasets. In addition, all measures of workload error include the same factor  $P(\epsilon, \delta)$ , so that changing the privacy parameters impacts each method with the same factor, leaving the ratio of their error the same. Consequently, for workload error, we simply fix  $\epsilon = 0.5$  and  $\delta = 0.0001$ .

For workload error, all error measurements are purely a function of the workload, reflecting the hardness of simultaneously answering a set of queries under differential privacy. In addition, these error rates can be compared directly with the lower bound as Theorem 2, reflecting a bound on the approximation rate. (This lower bound is not known to be achievable for all workloads, but nevertheless informs the quality of the eigen-strategy and its competitors.)

We also evaluate the relative error rates achievable using our algorithm by computing the strategy that minimizes absolute error on a scaled workload, as described in Sec. 3.4. Of course, the relative error rates reported in experiments are

always for the original input workload. In these experiments we vary the value of  $\epsilon$ , for a fixed  $\delta = 0.0001$ , and consider two real datasets. The first dataset is the US individual census data in the past five years<sup>3</sup>, which are aggregated on age, occupation and income. The second is the Adult dataset<sup>4</sup>, in which tuples are weight-aggregated on age, work, education and income. The size and dimensions of the datasets are:

Dataset	Dimension	# Tuples
US Census	$8 \times 16 \times 16$	15M
Adult	$8 \times 8 \times 16 \times 2$	33K

**Table 1: The size and dimensions of the datasets**

All experiments are executed on a quad-core 3.16GHz Intel CPU with 8 GB memory. Our Python implementation extends publicly-available code for the matrix mechanism [2] and also uses the `dsdp` solver [3] in the `cvxopt` [1] package.

### Competing Approaches

We compare the Eigen-Design strategy with the following four alternatives. Although originally proposed in the context of  $\epsilon$ -differential privacy, each is easily adapted to  $(\epsilon, \delta)$ -differential privacy and the shift generally improves the relationship to the optimal error rate (with the exception of the Fourier strategy, noted below).

**Fourier** is designed for workloads consisting of all  $k$ -way marginals, for given  $k$  [4]. The strategy transforms the cell counts with the Fourier transformation and computes the marginals from the Fourier parameters. When the workload is not full rank, the unnecessary queries of the Fourier basis are removed from the strategy to reduce sensitivity. The effectiveness of the Fourier strategy is somewhat reduced under  $(\epsilon, \delta)$ -differential privacy because dropping unnecessary queries results in a smaller sensitivity reduction using  $L_2$ .

**DataCube** is an adaptive method that supports marginal workloads [7]. We implemented the BMAX algorithm, which chooses a subset of input marginals so as to minimize the maximum error when answering the input workload. To adapt the algorithm to  $(\epsilon, \delta)$ -differential privacy, sensitivity is measured under  $L_2$  instead of  $L_1$ .

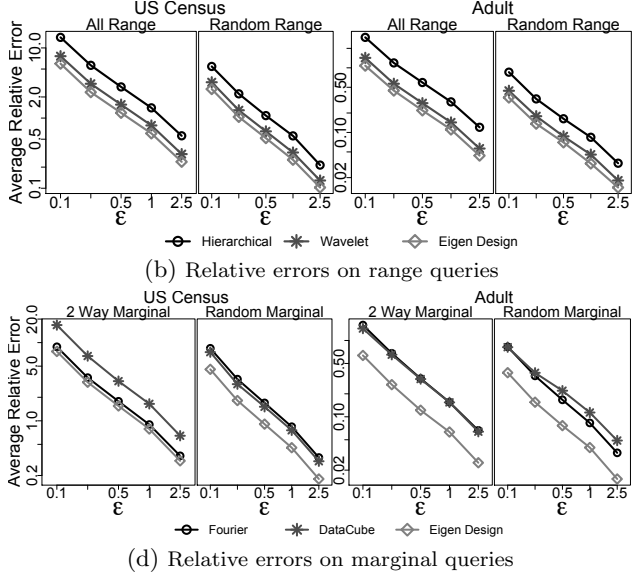
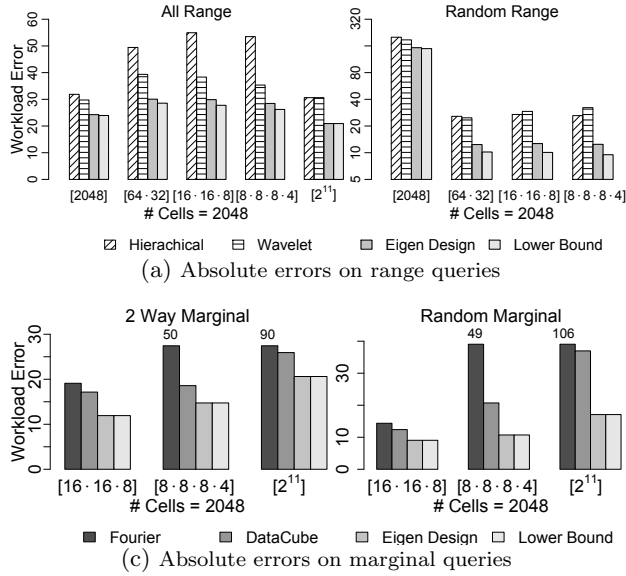
**Wavelet** supports multi-dimensional range workloads by applying the Haar wavelet transformation to each dimension [21]. When using  $\epsilon$ -differential privacy, Xiao et al. also introduced a hybrid algorithm that uses the identity strategy on dimensions with small size. This optimization is unnecessary under  $(\epsilon, \delta)$ -differential privacy: the hybrid algorithm does not lead to smaller error when sensitivity is measured under  $L_2$ .

**Hierarchical** aims to answer workloads of range queries using a binary tree structure of queries: the first query is the sum of all cells and the rest of the queries recursively divide the first query into parts [13]. We test binary hierarchical strategies (although higher orders are possible). The strategy in [13] supports one dimensional range workloads, but is adapted to multiple dimensions in a manner analogous to Wavelet [21].

We do not compare with the error of the standard Gaussian mechanism, which, for the workloads considered, is far

<sup>3</sup>Integrated Public Use Microdata Series: [usa.ipums.org](http://usa.ipums.org)

<sup>4</sup>UCI Machine Learning Repository: [archive.ics.uci.edu/ml/](http://archive.ics.uci.edu/ml/)



**Figure 3: Absolute and relative error for the Eigen-Design algorithm and competitors, for range and marginal workloads, on 2048 cells. “Lower Bound” is a bound on the best possible error achievable by any strategy.**

worse than all alternatives. Prior works [13, 21, 7] compared the error rates of their approaches with the identity strategy. We omit this explicit comparison, since the identity is always within the space of possible strategies the Eigen-Design could choose, but is not competitive.

### 5.1 Error of the Eigen-Design Algorithm

We now measure the improvement in absolute and relative error offered by the Eigen-Design algorithm along with its approximation to optimal absolute error. Below we refer to the strategy produced by the Eigen-Design algorithm, for a given workload, as the *eigen-strategy*. We consider three classes of workloads, beginning with workloads of range queries, then workloads of marginals, and then some alternative workloads designed to test the adaptivity of the mechanism.

**Workloads of Range Queries.** Figs. 3(a),(b) contain experiments on workloads of all range queries and random range queries. The random ranges are sampled with the two-step sampling method in [21]. Here the eigen-strategies are compared with Hierarchical and Wavelet strategy. The figures are in log scale, except Fig. 3(a) on all range queries. The results show that the eigen-design strategies reduce error by a factor of 1.2 to 2.1 in workload error and 1.3 to 1.5 in relative error compared to the best competing strategies. In addition, for workload error, the eigen-design strategy is within a factor of 1.3 to the lower bound.

**Workloads of Marginals.** Figs. 3(c),(d) contain experiments on workloads of 2-way marginal queries and random marginal queries, in which the random marginals are sampled with the sampling method in [7]. Here the eigen-strategies are compared with Fourier and DataCube. The figures are in linear scale for workload error and log scale for relative error. The results show that the eigen-design strategies reduce error by a factor of 1.3 to 2.2 compared to the best competing strategies in workload error, and by a factor of 1.1 to 2.7 in relative error. In addition, the error of eigen-design strategies match the lower bound of workload error,

indicating that our algorithm found an optimal strategy with respect to workload error.

Workload	Err Type	Error Ratio		Best/Worst Competitor
		Best/Worst	Bound	
1D Range (Permuted)	workload	9.62/13.16	0.99	Wav./Hier.
	relative	1.51/2.43	-	Wav./Hier.
1Way Range Marginal	workload	1.30/7.69	0.98	D.Cube/Four.
	relative	1.36/4.93	-	D.Cube/Four.
2Way Range Marginal	workload	1.63/3.23	0.95	Hier./Four.
	relative	1.81/2.38	-	Wav./D.Cube
1D CDF	workload	1.01/1.01	0.80	Wav./Hier.
	relative	0.46/0.54	-	Wav./Hier.
Predicate	workload	1.39/1.94	1.00	Wav./Four.
	relative	1.42/3.55	-	Four./Hier.

**Table 2: The factor of error reduced for the Eigen-Design algorithm w.r.t. the best/worst competitors strategies and the theoretical bound, for alternative workloads, on 2048 cells.**

**Alternative Workloads.** To demonstrate that our mechanism is adaptive over variety of workloads, we also include other workloads that have not been studied in prior work. First we show that our mechanism adapts to semantically equivalent workloads, in which we repeat the experiment on range Workload but randomly permute the order of cell conditions. The justification for this experiment comes from the fact that the user may wish to answer queries in which the order of the cell conditions is not obvious, such as predicate queries over categorical attributes.

In addition, we run experiments on three other workloads: the range marginals workload, the cumulative distribution (CDF) workload, and uniformly sampled predicate queries. The range marginals workload is important because most data analyses using marginals do not simply use individual counts, but also aggregate counts. If this is the case, simply computing the marginals workload privately is the wrong approach because error accumulates for aggregations. Last, the CDF workload is a highly-skewed set of one-dimensional range queries where the sensitivity in the first cell is  $n$ , decreasing linearly to 1 for the last cell.

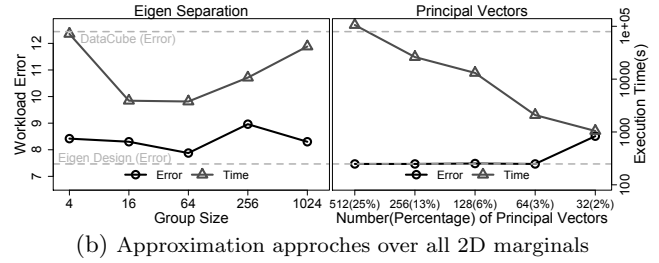
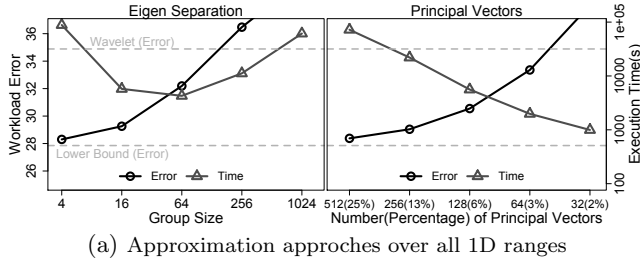


Figure 4: Quality and efficiency of approximation methods on 8192 cell conditions

We summarize the experimental results on alternative workloads in **Table 2**. For relative errors, due to space constraints, we only present results on US census data with  $\epsilon = 0.5$  and  $\delta = 0.0001$ . We present, for each workload, the factor of error reduction achieved by our algorithm compared to the best and worst competing approach, whose name is shown in the last column of the table. (Datacube is only considered for range marginals and Fourier is not considered on permuted range and CDF.) In addition, for workload error, we also include the ratio to the error lower bound.

The results show that the eigen-strategy can improve absolute error by as much as 13 times (on permuted range queries) and relative error as much as 5 times (on one-way range marginals). The workload error of competing strategies is heavily impacted by the permutation but the relative errors are not as bad since queries of individual cells and small ranges dominate the workload, which do not change too much under permutation. On all workloads but one, the eigen-strategy beats every competitor by at least a factor of 1.3, and is very close to—or achieves—the theoretical error lower bound. The only exception is the CDF workload, in which the eigen-strategy is only a bit better than the competitor for workload error and worse (than Hierarchical and Wavelet) for relative error. Overall, the results for workload and relative error are largely similar for range marginals and the predicate workload.

## 5.2 Performance Optimizations

**Fig. 4** illustrates the trade-off between computational speed-up and solution quality for the *eigen-separation* and *principal vector* performance optimizations described in Section 4. We only present results with workload errors here (the results with relative error are similar or even better). Error and computation time are plotted together using two y-axes: the left axis measures workload error and the right axis measures execution time in seconds. The baselines for error are the lower bound and the best competing technique.

The running time of using the standard Eigen-Design algorithm can be estimated from the running time of the principal vector method, which is more than an order of magnitude slower than the principal vector method with 25% of the eigenvectors. Comparing with this estimated time, both methods can reduce the running time by two orders of magnitude while the error they introduced is less than 12% over the lower bound. For the eigen-separation method, the computation in each group takes more time with larger group sizes while the computation of merging groups takes more time with smaller group sizes. Theoretically, the best choice for group size of the eigen-separation method is  $n^{1/3}$ , which is closest to 16 in this case. Using eigen-query separation with a group size of 16, the error is 5% higher on all range

queries and 11% higher on all marginal queries. Using the principal vectors optimization with 6% of the eigenvectors, the error is 10% higher on all range queries and the same as the optimal on all marginal queries.

According to the results, the eigen-separation performs better on range queries while the principal vectors method is better on marginals. In either case, the performance improvements still produce results that are significantly better than competing techniques.

## 5.3 The Choice of Design Queries

To evaluate our claim from Section 3.2 that the eigen-queries are an effective choice for the design queries we compare strategies computed by Program 1 using the eigen-queries, the Wavelet matrix and Fourier matrix as the design queries. Since using the eigen-queries introduces the same error to semantically equivalent workloads, we also empirically verify this property on other sets of designed queries. **Fig. 5** shows the results of those comparisons over two structured workloads considered above, as well as the same workloads with the order of the cell conditions permuted.

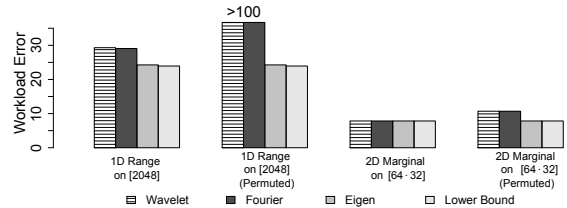


Figure 5: Comparison of design queries

The results show that using the Fourier or the Wavelet strategy as the set of design queries introduces 20% more error over all one dimensional range queries and achieves the same error on two-way marginals. However these design queries can not maintain their performance for workloads represented under a permutation of the cell conditions: they are worse than the eigen-queries by more than 4 times over the permuted one-dimensional range queries.

## 5.4 Experimental Conclusions

The experimental results show that, for the workloads specifically targeted by competing techniques, those techniques achieve error that is not too far from optimal (usually a factor of about 1.2 to 3.4 times the lower bound on error). But for broader classes or workloads, or ad hoc subsets of structured workloads, existing techniques are limited and the adaptivity of the Eigen-Design can improve relative or absolute error by a larger factor. We have confirmed the versatility of our algorithm, as it improves on all competing techniques for virtually every workload considered. The one

exception is the highly skewed CDF workload. The lowest error strategy we are aware of for this workload is produced by our design algorithm, but with an alternative basis.

## 6. RELATED WORK

The present work uses the framework of the matrix mechanism to develop an adaptive query answering algorithm. The original work on the matrix mechanism [14] described and analyzed in a unified framework two prior techniques specifically tailored to range queries. The first used a wavelet transformation [21]; the second used a hierarchical set of queries followed by inference [13]. Originally, the matrix mechanism focused mainly on  $\epsilon$ -differential privacy, although  $(\epsilon, \delta)$ -differential privacy was also considered briefly. Prior work on the matrix mechanism never considered strategies beyond those proposed in the previous literature, or natural candidates like the identity matrix. The convex optimization formalization in prior work only runs on small  $n$  ( $n < 64$ ) and cannot be used in practice.

Low order marginals are studied in [4] using Fourier transformation. They also consider enforcing integral consistency on the output, an objective we do not consider here. Recently, Ding et al. proposed an adaptive algorithm to answer workloads consisting of data cube queries [7], which (described in our terms) considers strategies composed only of individual marginal queries and optimizes the workload error approximately. The algorithm adapts a known approximation algorithm for the subset-sum problem and cannot be applied to general linear queries. Most of these techniques focus on  $\epsilon$ -differential privacy, however they are actually more effective under  $(\epsilon, \delta)$ -differential privacy, so comparisons with our algorithms are meaningful.

The error rates of the matrix mechanism are independent of the database instance. Recently, a number of data dependent algorithms for answering linear queries under differential privacy have been proposed. Xiao et al. [22] propose a method for computing a strategy matrix using KD-trees, and Cormode et al. [6] propose a related method in which a differentially-private median computation is used to guide hierarchical range queries. While promising, these approaches appear to restrict the strategy to hierarchical structures which we have shown are suboptimal for many workloads. Dynamic strategy selection can also increase computation cost. These tradeoffs deserve further investigation.

Focusing on relative error, Xiao et al. [20] propose a data-dependent algorithm to minimize the relative error with an innovative resampling function. Data-dependent interactive (as opposed to batch) mechanisms have been considered by Roth and Roughgarden [19], who answer predicate queries on databases with 0-1 entries. Hardt et. al [12] provide a linear time algorithm for the same query and database setting.

## 7. CONCLUSIONS AND FUTURE WORK

We have described an adaptive mechanism for answering complex workloads of counting queries under differential privacy. The mechanism can be seen to automatically select, for a given workload, a noise distribution composed of linear combinations of independent Gaussian noise. With no reduction in privacy, the mechanism can significantly reduce error over competing techniques and is close to optimal with respect to the class of perturbation methods considered.

In the future we hope to extend our theoretical approximation bounds to the eigen-separation and principal vector optimizations, and apply our approach to non-linear queries.

**Acknowledgements.** Li and Miklau were supported by the NSF through grants CNS-1012748 and IIS-0964094. We are grateful for the comments of the anonymous reviewers.

## 8. REFERENCES

- [1] <http://abel.ee.ucla.edu/cvxopt/>.
- [2] <http://dbgroupp.cs.umass.edu/code/>.
- [3] <http://www.mcs.anl.gov/hs/software/dsdp/>.
- [4] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency: A holistic solution to contingency table release. In *PODS*, pages 273–282, 2007.
- [5] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.
- [6] G. Cormode, M. Procopiuc, E. Shen, D. Srivastava, and T. Yu. Differentially private spatial decompositions. In *ICDE*, 2012.
- [7] B. Ding, M. Winslett, J. Han, and Z. Li. Differentially private data cubes: optimizing noise sources and consistency. In *SIGMOD*, pages 217–228, 2011.
- [8] C. Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54(1):86–95, 2011.
- [9] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *EUROCRYPT*, volume 4004/2006, pages 486–503, 2006.
- [10] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, volume 3876/2006, pages 265–284, 2006.
- [11] A. Ghosh, T. Roughgarden, and M. Sundararajan. Universally utility-maximizing privacy mechanisms. In *STOC*, pages 351–360, 2009.
- [12] M. Hardt and G. Rothblum. A multiplicative weights mechanism for privacy-preserving data analysis. In *FOCS*, pages 61–70, 2010.
- [13] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *PVLDB*, 3(1-2):1021–1032, 2010.
- [14] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. In *PODS*, pages 123–134, 2010.
- [15] C. Li and G. Miklau. Measuring the achievable error of query sets under differential privacy. *CoRR*, abs/1202.3399, 2012.
- [16] F. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *SIGMOD*, pages 89–97, 2009.
- [17] F. McSherry and I. Mironov. Differentially private recommender systems: Building privacy into the netflix prize contenders. In *SIGKDD*, pages 627–636, 2009.
- [18] F. Pukelsheim. *Optimal design of experiments*. Wiley-Interscience, 1993.
- [19] A. Roth and T. Roughgarden. Interactive privacy via the median mechanism. In *STOC*, pages 765–774, 2010.
- [20] X. Xiao, G. Bender, M. Hay, and J. Gehrke. iReduct: Differential privacy with reduced relative errors. In *SIGMOD*, pages 229–240, 2011.
- [21] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. In *ICDE*, pages 1200–1214, 2010.

- [22] Y. Xiao, L. Xiong, and C. Yuan. Differentially private data release through multidimensional partitioning. In *SDM*, pages 150–168, 2010.